(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 478 175 A1**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number : **91308276.4**

(51) Int. Cl.^{5} : **H04L 12/26**

(22) Date of filing : **11.09.91**

(30) Priority : **13.09.90 EP 90310017**

(43) Date of publication of application : **01.04.92 Bulletin 92/14**

(84) Designated Contracting States : **DE FR GB IT**

(71) Applicant : **Hewlett-Packard Company
3000 Hanover Street
Palo Alto, California 94304 (US)**

(72) Inventor : **I'Anson, Colin Stephen
1 Fabian Drive
Stoke Gifford, Bristol BS12 6XL (GB)**
Inventor : **McKee, Neil
27 Longmead Avenue
Horfield, Bristol (GB)**
Inventor : **Galloway, James Robertson
Jahn Strasse 16
W-7038 Holzgerlingen (DE)**

(74) Representative : **Squibbs, Robert Francis et al
Hepworth, Lawrence, Bryer & Bizley, Lewins House, Lewins Mead
Bristol BS1 2NN (GB)**

(54) A protocol analyzer.

(57) A protocol analyzer is provided for monitoring a selected communication connection being conducted in accordance with a predetermined protocol by the exchange of protocol data units between two entities over a data network (12). The analyzer (10) comprises monitoring means (13-15) for identifying protocol data units, a protocol-follower (17,18,27) conditioned in dependence on said predetermined protocol and operative to follow the progress of the connection as it receives relevant protocol data units from the monitoring means (13-15), and alarm means (22) indicating when the sequence of protocol data units diverges from the protocol. The protocol data units are passed through a FIFO store (16) as they are used by the protocol-follower, so that on a protocol violation occurring, the sequence of protocol data units leading up to the violation can be extracted from the FIFO store (16) and displayed. The protocol analyzer thus filters out protocol data units conforming to the relevant protocol so that only protocol data units violating the protocol, together with the immediately preceding protocol data units, are displayed.

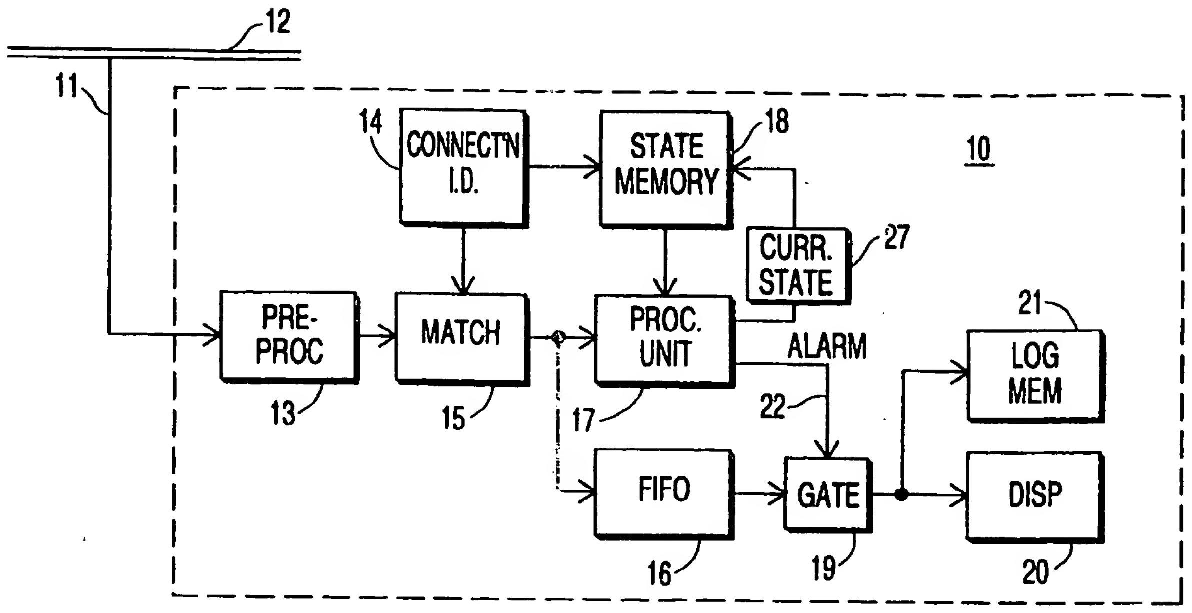

12
11
10
14
CONNECTN I.D.
STATE MEMORY
18
CURR. STATE
27
PRE-PROC
13
MATCH
15
PROC. UNIT
17
ALARM
22
21
LOG MEM
FIFO
16
GATE
19
DISP
20

FIG 4

The present invention relates to a protocol analyzer for monitoring a selected communication connection being conducted in accordance with a predetermined protocol between two entities over a data network.

A data network is a network (which may or may not include nodes performing switching functions) interconnection a plurality of data processing devices. Such networks are often used to interconnect a number of computers, but can also be used for other data communication purposes, such as telephone-type networks.

Information is generally transmitted in the form of messages, and an individual message will often be divided into a number of discrete packets. In such a network, the routing of the packets may be at least partially controlled or determined by the various nodes in the network. In some cases, the route taken by the packets of a message is set up for a connection and all packets follow that route; in others, the various packets for a connection may follow different routes through the network. Usually, a number of packets of different connections will be interleaved on any particular link between two adjacent nodes in the network.

Such networks are liable to suffer from faults. The cause and/or effect of a fault may both be immediately evident. For example, the physical linkage between two nodes may be interrupted; or no packets may be received from a particular node. However, network faults are often subtle in both their causes and effects, and it may even not be clear whether there is a fault or not; for example, a poor response time of the network may be due to a fault or it may be due to an unusual and extreme workload imposed on it.

A variety of instruments are available for network fault diagnosis (using the term "diagnosis" in a broad sense). At the lowest level, there are voltage level testers, continuity testers, etc. At a slightly higher level, there are signal presence testers such as LED instruments.

However, many network faults occur at a higher level and take the form of a violation of the operating procedure, or protocol, which the communicating entities have agreed (generally implicitly) will govern their communication. Matters are further complicated by the fact that when, for example, two computers communicate over a network, they generally do so using a multi-layered protocol stack to format and control the communication process. In such cases, at least conceptually, a protocol entity of each layer of the protocol stack of one computer communicates with a corresponding protocol entity of the same layer of the other computer, this communication being by the exchange of so-called "protocol data units" that carry control information relevant to the protocol layer concemed as well as user data. Thus, several protocols, each at a different level, are concurrently in use, any one of which could give rise to a communication fault.

Because a protocol is describable in terms of a set of states with transitions between the states being determined, at least in part, by the type of protocol data unit received (this type being set by the control information of the protocol data unit), it is possible to identify protocol violations by looking at the sequence of protocol data units passing over the network in respect of the relevant pair of communicating entities. A protocol analyzer is an instrument designed to carry out such a task.

In using a protocol analyzer, the analyzer is attached to a suitable point in the network. It detects and analyses packets to produce a listing of the protocol data units relevant to the protocol being monitored (generally a low-level, e.g link level protocol). By inspecting this listing, the operator can see the nature of the traffic at the point and recognize various kinds of errors.

The simplest form of listing is a listing of all protocol data units associated with the protocol being monitored. However, analysis of such a "raw" listing is an onerous task. There will usually be a large variety of sets of such protocol data units passing any given point in the network, each set having a different source and/or destination entity. Further, both the number and the complexity of the protocols used in a typical large system are large.

It has therefore become known to make the data collection by protocol analyzers "programmable". That is, the analyzer can be set by the user to respond only to conditions determined by the user. These conditions can be regarded as a filter which operates on the input data stream, and can be described as "mask" filtering.

The user can thereby select only protocol data units passing between two selected protocol entities. The user can also set the analyzer to respond only to certain sequences of protocol data units; obviously, the chosen sequences will usually be those indicative of errors or abnormal conditions. As just noted, however, the complexity of a protocol can often be high, and the number of protocols or variations thereof is also liable to be high. Thus programming the analyzer is an arduous task, and the user is quite likely to program it so as to detect only a limited number of "likely" abnormal conditions, and rely on a continuous scrolling display of all protocol data units to detect other abnormal conditions.

It is an object of the present invention to provide a protocol analyzer that facilitates the task of detecting protocol violations.

According to the present invention, there is provided a protocol analyzer for monitoring a selected communication connection being conducted in accordance with a predetermined protocol between two entities by the exchange of protocol data units over a data network, said protocol being a cyclically-operating, multiple-state protocol state-to-state transitions of which are caused by differing types of protocol data units, and said network being of the type that can support a plurality of simultaneous communication connections between different pairs of entities, said analyzer comprising:

- **monitoring means** for monitoring the network to derive protocol-unit signals indicative of protocol data units relevant to said selected connection;
- **protocol-follower means** including **current-state memory means** for storing an indication of the current protocol state of said selected connection, said protocol-follower means being conditioned in dependence on the states of said protocol, what types of protocol data units are validly received in each state, and any resulting change in state caused by each validly-received protocol data unit, to follow the execution of said protocol in relation to said selected connection, by examination of each said protocol-unit signal with reference to the current protocol state as indicated by said current-state memory means, said protocol-follower means being operative to update said indication of the current protocol state as appropriate; and
- **alarm means** operatively associated with said protocol-follower means and arranged to generate an output indication when a said relevant protocol data unit, as represented by the corresponding protocol-unit signal, is an invalid one for the current protocol state.

Due to this arrangement, all protocol violations will be detected without each possible violation having to be separately programmed into the protocol analyzer and without the user being required to monitor a listing of protocol data units the vast majority of which conform to the protocol.

The protocol-follower means and alarm means together form what may be termed a "context" or "behavioural" filter.

Preferably, the protocol-follower means is conditioned in dependence on said protocol as considered in terms of the possible states of the overall connection having regard solely to protocol data units exchangeable between the entities. However, it is also possible for the protocol-follower to be conditioned in dependence on said protocol as considered at the communicating entities, the protocol-follower means being operative to track the progress of the connection of each communicating entity separately with the current protocol state of each entity being held in the current-state memory means.

Generally, the relevant protocol information will be stored in a state memory in the protocol-follower means although it would also be possible to provide dedicated hardware emulating a protocol state machine. In the former case, the state memory advantageously comprises a two-dimensional look-up table accessed by current state and protocol-data-unit type as indicated by said protocol-unit signal, said table having for each state, a valid-unit entry for each protocol-data-unit type validly received in that state and an invalid-unit entry for invalid protocol-data-unit types, each said valid-unit entry indicating the protocol state following receipt of that protocol-data-unit type concerned.

The protocol analyzer may include output means controlled by the alarm means to generate a protocol-data-unit-representing output only for each protocol data unit indicated as invalid by the alarm means. Alternatively, the output means can be arranged to generate a protocol-data-unit-representing output for each said relevant protocol data unit, the output means being controlled by the alarm means to distinguish invalid protocol data units from valid protocol data units. As a further alternative, the output means may operate to generate a protocol-data-unit-representing output for each of a limited number of valid relevant protocol data units that immediately precede an invalid protocol data unit as well as for the invalid protocol data unit itself.

In order to permit the user to change what is considered a violation (for example, where the protocol actually in use is a variant of the one conditioning the protocol-follower), the conditioning of said protocol-follower means is preferably user modifiable.

Advantageously, the protocol-follower means is so conditioned as to permit the user to select predetermined protocol data units that are otherwise valid in particular protocol states, for identification by said alarm means when occurring in those states (these predetermined units, are, for example, ones that are unusual, though valid, in a particular state). Operation without this feature would result in the detection of only strictly protocol-violating events ("strong filtering"), while operation with the feature selected would result in the automatic detection of the permitted but unusual sequences as well ("weak filtering"). The permitted but unusual sequences could of course be indicated in a manner distinguishing them from the strictly forbidden sequences.

A protocol analyzer embodying the present invention will now be described, by way of non-limiting example, with reference to the accompanying diagrammatic drawings, in which:

Figure 1 is a diagram illustrating protocol stacks of two communicating end systems;

Figure 2 is a diagram illustrating an allowed sequence of protocol-data-unit types exchangeable between corresponding protocol entities according to a pre-determined protocol;

Figure 3 is a state transition diagram for the protocol associated with Figure 2, considered from the point of view of the overall connection established between the communicating protocol entities;

Figure 4 is a block diagram of the protocol analyzer embodying the invention;

Figure 5 is a more detailed block diagram of a protocol state memory and processing unit of the Figure 4 protocol analyzer; and

Figure 6 is a state transition diagram for the protocol associated with Figure 2, considered from the point of view of the two communicating protocol entities.

Figure 1 of the accompanying drawings shows conceptual protocol stacks 1 and 2 operated by two end systems in communicating with each other over a network (12). Each protocol stack 1,2 is made up of a number of different layers each of which performs a particular task in the communication process. Considering by way of explanation, layer N in each protocol stack 1,2, this layer N provides services to the layer above (layer N+1) and in doing so utilizes services provided by the layer below (layer N-1).

Within each layer N a protocol entity 3,4 controls the carrying out of the communication tasks assigned to that layer, this control being effected in coordination with the corresponding protocol entity of the communicating end system. Conceptually the protocol entities 3,4 in the same protocol layer of the communicating end systems communicate and coordinate with each other in accordance with a peer protocol (for layer N this is the layer N protocol shown in Figure 1). The peer protocol defines the form and sequencing of messages passed between the peer protocol entities 3,4 in the form of protocol data units 5,6. Each protocol data unit (PDU) 5,6 contains protocol control information PCI and one or more service data units SDU, the latter being data which the layer N protocol entity is handling on behalf of the layer N+1 above.

Whilst conceptually the peer protocol entities 3,4 are communicating with each other by passing protocol data units directly between themselves, in practice, of course, the protocol data units must pass down one protocol stack, across the network 12 and up the other protocol stack to the relevant layer N. A protocol data unit passed by the protocol entity of layer N down to layer N-1 is treated by that latter layer as a service data unit SDU and handled appropriately.

Such conceptual layering of communication protocol stacks is well known in the art - see, for example, the seven-layer OSI (Open Systems Interconnect) model defined by the International Standards Organization (International Standard ISO7498). It will be appreciated that in practice the protocol stacks are implemented primarily in software, although the lower levels may well be effected using dedicated hardware.

A protocol entity can be arranged to handle multiple connections simultaneously, each connection being used to transfer data to/from a different end-system process. In this case, it is, of course, necessary for each connection to be uniquely identified so that the entity knows which protocol data unit relates to which connection. By way of example, protocol entities running the well known transmission control protocol TCP, use pairings of endpoint to identify a connection, where an endpoint is the combination of a parameter (IP address) identifying the end system concerned (or, more accurately, an interface of the end system to the network), and a parameter (port number) indicating the source/destination within the end system with which the TCP protocol entity is to communicate. A fuller explanation of TCP connection identification may be found in a reference work such as "Internetworking with TCP/IP", Douglas E. Comer, Volume 1, Second Edition 1991, Prentice-Hall.

From the foregoing, it can be seen that even when only a single pair of end systems intercommunicate, a number of different protocols are being simultaneously operated at different levels between pairs of protocol entities, each such pair effectively conducting one or more communication connections.

The peer protocol controlling communication between corresponding protocol entities will often specify that certain types of protocol data units (as defined by the protocol control information carried by the units) must be preceded and/or followed by certain types of other protocol data units. Figure 2 illustrates for a simple protocol a typical allowable sequence of protocol data units types. More particularly, in response to a request " Open" from the protocol layer N+1 to establish a communicating connection, the protocol entity 3 sends out a connection request (CR) via a corresponding protocol data unit to its peer protocol entity 4 in the end system with which it is desired to establish communication. On receiving this request, the peer entity 4 returns a connection confirm (CC) message (assuming that the entity 4 is ready to receive data). Thereafter, and while data is available to the entity 3 (as indicated by a signal "dav"). The protocol entity 3 will send data in protocol data units to the protocol entity 4. In due course, no more data is available and the protocol entity 3 sends a disconnect request (DR) and the entity 4 responds with a disconnect confirm (DC) message.

The protocol governing communication between the entities 3 and 4 can be viewed in terms of the allowable sequences of protocol data units that can appear on the network 12. Figure 3 is a state transition diagram for the protocol used for the Figure 2 communication viewed this way. The Figure 3 state transition diagram comprises five states S1 to S5 with transition between the states occurring in correspondence to particular types of protocol data units being transferred across a network. It will be appreciated that this view of the protocol is very much a connection view as seen from the network and is a passive one (i.e. no actions are consequent on a change of state). More particularly, initially no connection is present between the protocol entities 3 and 4 and the connection is effectively in its Idle State S1. In due course, one protocol entity will seek to establish communication with a peer protocol entity and a communication request (CR) will appear on the network moving the overall connection to an "Awaiting CC" state S2 in which the connection is next expecting to see a connection confirm (CC) protocol data unit. When such a protocol data unit arrives, the connection moves into a "ready" state S3 indicating that both the protocol entities 3,4 are ready for the transfer of data. In due course, a first data-bearing protocol data unit is passed over the network and the "transfer data" state S4 is accordingly entered. As further data is transferred, the connection remains in its transfer data state S4. When data transfer is complete a disconnect request (DR) will appear on the network and the connection moves into its "Awaiting DC " state S5 in which the next expected message will be a disconnect confirm one. When this latter message passes over the network, the connection moves back into its Idle State S1.

The Figure 3 state transition diagram also provides for one further transition, this being a transition from the ready state 53 directly to the "Awaiting DC" state S5 in response to the receipt of a disconnect request (DR) message before any data has been transferred. This transition is shown in dashed lines in Figure 3 because although it is an allowable transition, it is unusual for a connection to be closed down without the transfer of any data.

It will be appreciated that the protocol represented by Figure 3 is a very simple one and that, in fact, the state transition diagram does not adequately deal with all eventualities; however, the protocol described above with reference to Figures 2 and 3 is only illustrated for the purposes of explaining the operation of the protocol analyzer of the invention and does not, of course, form any part of the present invention.

As already noted, each protocol entity may, in fact, be running more than one communication connection at any one time with each such connection being at a different state in its protocol cycle. If this is the case the protocol control information part of each PDU will contain sufficient information to associate the protocol data unit with a particular connection being run by the protocol entity.

Referring now to Figure 4, the protocol analyzer 10 embodying the invention is coupled at 11 to a channel 12 in the data network being monitored (obviously the analyzer can be attached instead to a node of the network). The analyzer is thus fed with the packets passing along the channel 12.

Each packet on the channel 12 is picked off and fed to a pre-processing unit 13, which identifies and separates the various components of the packet, including, in particular, connection-identifying parameters. The identifying parameters of the connection to be monitored are set by the user in a store 14, these parameters including end-system pairing, protocol entities involved, and a parameter for distinguishing between multiple connections involving the same entities. The stored connection-identifying parameters are matched against those derived by the unit 14 in a match unit 15 to identify PDUs relevant to the connection of interest. The match unit 15 passes on the type of each relevant PDU to a FIFO store 16.

The PDU types passing through the unit 15 are also fed to a protocol-follower constituted by a processing unit 17, a current state memory 27 for holding the current state of the protocol relevant to the connection under consideration, and a protocol state memory 18 holding data that indicates for each state of the relevant protocol, what types of PDU are validly received when that state is the current state, and what state transition, if any, is caused by such receipt. Where several protocols are being simultaneously operated (for example, peer protocols of different layers of the protocol stacks of two communicating end systems), then the part of the memory 18 holding data on the protocol relevant to the connection being monitored, is selectively enabled by an appropriate enabling signal generated by the connection identification unit 14.

For each PDU type passed on by the match unit 15, the processing unit 17 determines by reference to the data held in the memory 18, whether the type is a valid one for the current protocol state. Where the PDU type is not a valid one, the unit 17 generates an alarm signal on line 22 which enables a gate 19, allowing the sequence of PDU types leading up to the protocol violation, as well as the violating PDU type, to pass to a display unit 20 and a log store 21. If the PDU is a valid one, the processing unit 17 updates the memory 27 to hold the new current protocol state of the connection.

Sequences violating the protocol are thus displayed to the user, and stored in unit 21 for later analysis if desired. The user can see not just the protocol-violating PDU type but the whole sequence leading up to it, because on a protocol violation, that whole sequence is contained in the store 16. The gate 19 is of course kept enabled while the whole sequence passes through it.

Figure 5 is a block diagram of the protocol-follower shown as units 17, 18 and 27 in Figure 1, though the division into those units is functional; as will be seen, the various components illustrated in Figure 5 that go to make up the units 17 and 18 cannot readily be separated. By way of example, the protocol-follower will be described below in relation to the protocol illustrated in Figures 2 and 3.

The protocol state transition table (Figure 3) is effectively stored in a matrix store 25. In this store, the columns represent the states (S1 to S5), and the rows represent the PDU types (CR, CC, DATA, DR, and DC) which cause state transitions. For each possible transition (from a current state to a next state in response to a PDU type), the matrix store 25 contains the next state code in the element at the intersection of the column for the current state and the row for the PDU type. Thus the transition from state S3 induced by a PDU type DATA is to state S4, and is stored by the code S4 in the element at the intersection of column S3 and row DATA. All other matrix store locations contain null entries.

The incoming PDU type from unit 15 is fed to a row decoder 26 to select the appropriate row of the matrix 25. Also, the current state stored in the current state memory 27 is passed to a state decoder 28 to select the appropriate column of the matrix 25. If the element so selected contains a state code, that code is read out and passed to a next state store 28. The fact that that element contains a state code indicates that the PDU type is a valid one. If however the selected element does not contain a state code, the next state store 29 will remain empty. This condition, which is caused by an invalid PDU type, is indicated by an alarm signal on the line 22.

After the matrix store 25 has been read and the new state (if any) read into the next state store 29, the contents of the next state store 29 are transferred to the current state memory 27 (so erasing its previous contents) in readiness for the reception of the next PDU type.

The circled entry for state S3 and row DR is the permitted but abnormal transition mentioned above. If desired, this entry may have a distinguishing code attached to it, with the output of the matrix 25 being fed to the next state store 29 via a code rejection unit 30 which can be enabled or disabled at the option of the user. If disabled, the unit 30 will allow all state values to pass; if enabled, it will allow only state values without a distinguishing code to pass. This implements the strong and weak filtering option described above.

Also, means may be provided whereby the user can enter and store selected transitions (i.e. selected PDU type sequences), which are then inhibited from being recognized as violations, and/or suppress selected transitions, which are then treated as violations. This may be done by providing a user-modifiable matrix which is operated in parallel with the main protocol state table (matrix store 25).

The gate 19 can of course be arranged to allow all packet types emerging from the FIFO store 16 and merely add a distinguishing tag to protocol-violating packet types, with the display unit 20 displaying tagged packet types in a distinctive manner.

As already noted, the state transition diagram of Figure 3 views the protocol governing the connection of interest, strictly in terms of the connection across the network and what PDU types pass over the network. It is also possible to specify the protocol considered from the point of view of the protocol entities themselves, although such a specification will generally involve primitives passed between protocol layers as well as the PDU types. Thus, Figure 6 illustrates the protocol associated with Figures 2 and 3 in terms of a state transition diagram as implemented by the communicating protocol entities. In this diagram, the PDU types (CR, CC, DATA, DR, DC) and primitives ('open','dav') causing state transitions are shown on the arrows between states, whilst the actions initiated by a state transition are shown in square brackets.

As can be seen, the Figure 6 state transition diagram comprises six states S10-S15, one of which (state S10) is an Idle State, three of which (S11-S13) are associated with data transmission, and the remaining two of which (S14, S15) are concerned with data reception. As the Figure 6 diagram is self-explanatory and will be readily understood by persons skilled in the art, a detailed textual description will not be given. The main point to note is that the diagram differs from that of Figure 3 in certain aspects.

The protocol analyzer can be arranged to follow the progress of a connection in terms of a state transition diagram of the Figure 6 form rather than of the Figure 3 form. In this case, although the matrix store 25 need only contain one copy of the relevant state transition data, it is generally advantageous for the analyzer to separately follow the progress of the connection at the two communicating entities so that the current state memory 27 must now hold two current states, one for each entity. Furthermore, each relevant PDU now has two different aspects: firstly, it serves to indicate that the sending protocol entity has carried out an action from which it can generally be deduced that it has passed from its current state into another state, and secondly, it serves to indicate the stimulus about to be received by the receiving protocol entity and therefore what state transition that entity is about to undergo. However, because PDU type is not the sole determinant of state transitions, there may well be ambiguity existing at least for a limited period, when PDU type alone is used to follow a complex protocol represented in terms of a state transition diagram as viewed by a protocol entity. Accordingly, it is preferred to use a diagram that relies solely on PDU type (such as that of Figure 3) when implementing the protocol analyzer of the invention.

Claims

1. A protocol analyzer for monitoring a selected communication connection being conducted in accordance with a predetermined protocol between two entities (3,4) by the exchange of protocol data units over a data network (12), said protocol being a cyclically-operating, multiple-

state protocol state-to-state transitions of which are conditioned by differing types of protocol data units, and said network (12) being of the type that can support a plurality of simultaneous communication connections between different pairs of entities, said analyzer comprising:

- **monitoring means** (13,14,15) for monitoring the network (12) to derive protocol-unit signals indicative of protocol data units relevant to said selected connection;
- **protocol-follower means** (17,18,27) including **current-state memory means**(27) for storing an indication of the current protocol state of said selected connection, said protocol-follower means being conditioned in dependence on the states (51-55) of said protocol, what types of protocol data units are validly received in each state, and any resulting change in state caused by each validly-received protocol data unit, to follow the execution of said protocol in relation to said selected connection, by examination of each said protocol-unit signal with reference to the current protocol state as indicated by said current-state memory means (27), said protocol-follower means (17,18,27) being operative to update said indication of the current protocol state as appropriate; and
- **alarm means** (22) operatively associated with said protocol-follower means (17,18,27) and arranged to provide an output indication when a said relevant protocol data unit, as represented by the corresponding protocol-unit signal, is an invalid one for the current protocol state.

2. A protocol analyzer according to claim 1, wherein said protocol-follower means (17,18,27) is conditioned in dependence on said protocol as considered in terms of the possible states (51-55) of the overall connection having regard solely to protocol data units exchangeable between the entities (3,4).

3. A protocol analyzer according to claim 1, wherein said protocol-follower means (17,18,27) is conditioned in dependence on said protocol as considered at the communicating entities (3,4), the protocol-follower means being operative to track the progress of the connection of each communicating entity separately with the current protocol state of each entity being held in said current-state memory means (27).

4. A protocol analyzer according to any one of the preceding claims, wherein said protocol-follower means further includes:

- **state memory means** (18) for storing for each state of said protocol, state data indicating what types of protocol data units are validly received in that state and any resulting change in state caused by each validly-received protocol data unit, and
- **processing means** (17) operative to follow the execution of said protocol, by examination of each said protocol-unit signal with reference to said state data associated with the current protocol state.

5. A protocol analyzer according to claim 4, wherein said state memory means (18) comprises a two-dimensional look-up table (25) accessed by current state and protocol-data-unit type as indicated by said protocol-unit signal, said table having for each state (51-55), a valid-unit entry for each protocol-data-unit type validly received in that state and an invalid-unit entry for invalid protocol-data-unit types, each said valid-unit entry indicating the protocol state following receipt of that protocol-data-unit type concerned, and said processing means (17) being operative to access said table (25) in dependence on the current protocol state and the type of the protocol data unit under consideration, whereby to ascertain whether that protocol-data-unit type is valid for the current state and, if so, the identity of the next protocol state.

6. A protocol analyzer according to claim 1, including output means (20) controlled by said alarm means (22) to generate a protocol-data-unit-representing output only for each protocol data unit indicated as invalid by said alarm means.

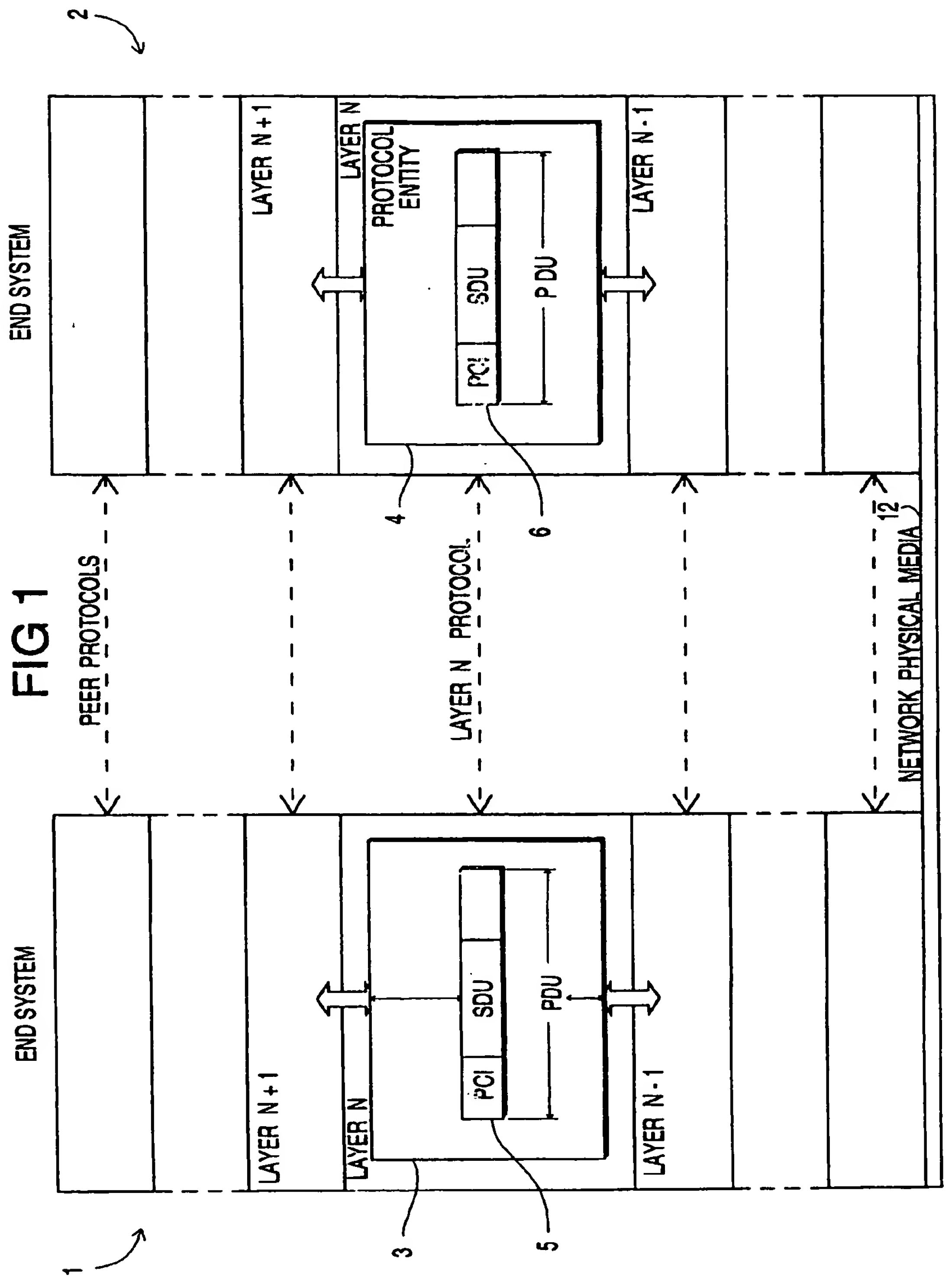
7. A protocol analyzer according to claim 1, including output means (20) operative to generate a protocol-data-unit-representing output for each said relevant protocol data unit, the output means (20) being controlled by said alarm means (21) to distinguish invalid protocol data units from valid protocol data units.

8. A protocol analyzer according to claim 1, including output means (20) controlled by said alarm means (22) and operative to generate a protocol-data-unit-representing output for each of a limited number of valid relevant protocol data units that immediately precede an invalid protocol data unit as well as for the invalid protocol data unit itself.

9. A protocol analyzer according to any one of the preceding claims, wherein the conditioning of said protocol-follower means (17,18,27) is user modifiable.

10. A protocol analyzer according to any one of the preceding claims, wherein the protocol-follower means (17,18,27) is so conditioned as to permit the user to select predetermined protocol data units that are otherwise valid in particular protocol states, for identification by said alarm means when occurring in those states.

FIG 1


END SYSTEM
LAYER N + 1
LAYER N
PROTOCOL ENTITY
PCI
SDU
P DU
LAYER N - 1
END SYSTEM
LAYER N + 1
LAYER N
PCI
SDU
PDU
LAYER N - 1
PEER PROTOCOLS
LAYER N PROTOCOL
NETWORK PHYSICAL MEDIA
1
2
3
4
5
6
12

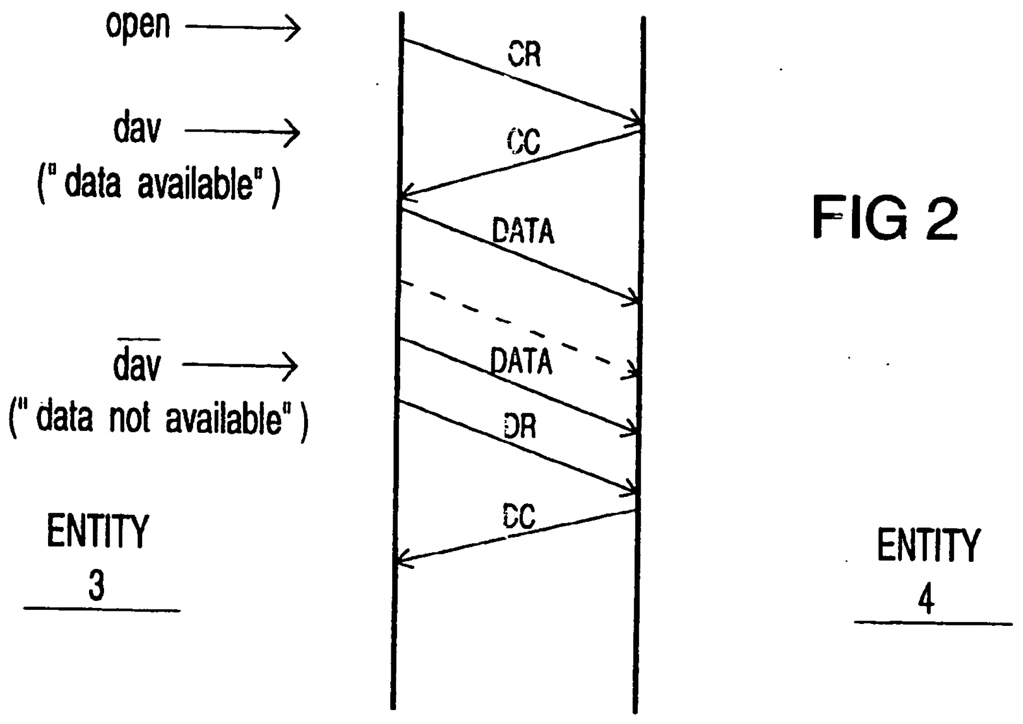

open
dav
(" data available")
$\overline{\text{dav}}$
(" data not available")
CR
CC
DATA
DATA
DR
DC
ENTITY 3
ENTITY 4

FIG 2

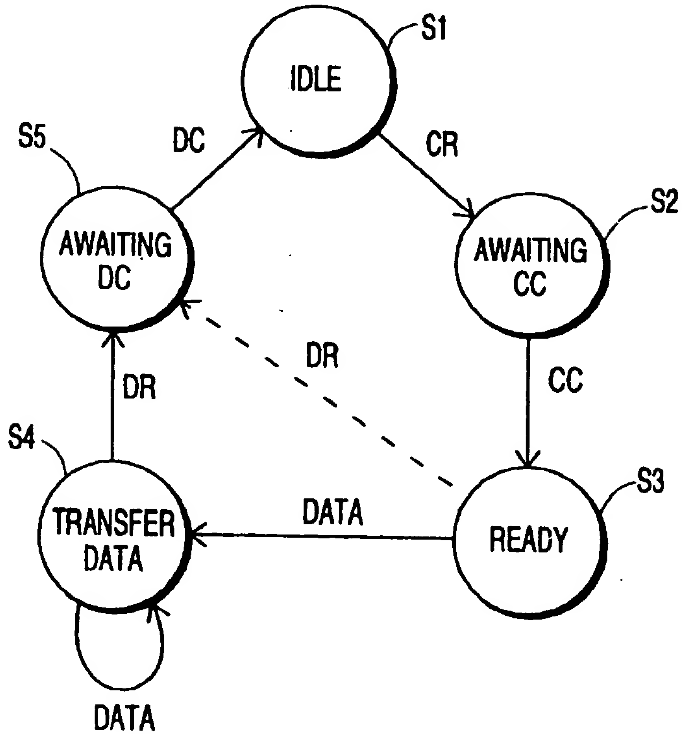

IDLE
S1
CR
AWAITING CC
S2
CC
READY
S3
DATA
TRANSFER DATA
S4
DATA
DR
AWAITING DC
S5
DC
DR

FIG 3

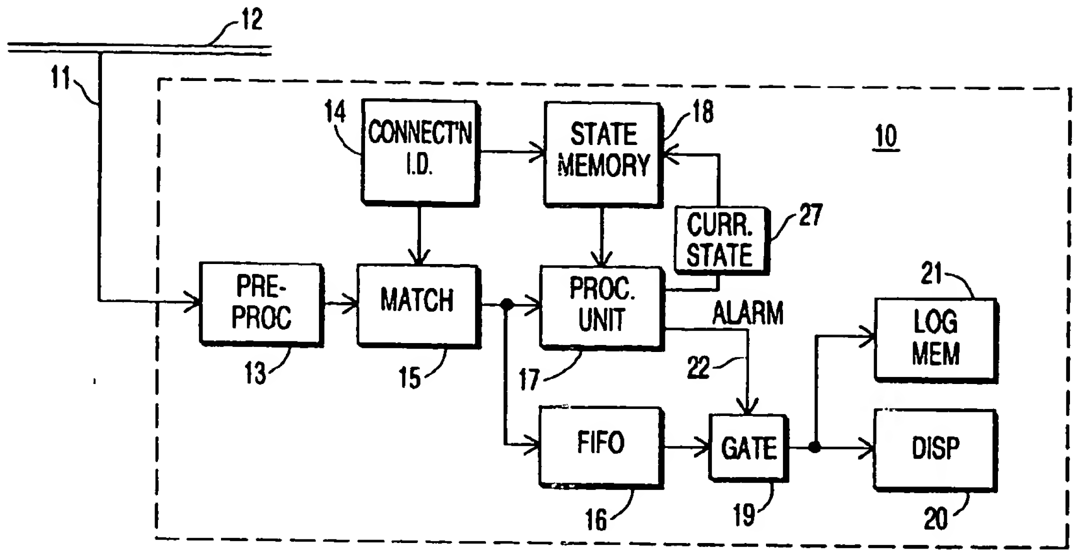

12
11
14
CONNECT'N I.D.
STATE MEMORY
18
10
CURR. STATE
27
PRE-PROC
MATCH
PROC. UNIT
ALARM
21
LOG MEM
13
15
17
22
FIFO
GATE
DISP
16
19
20

FIG 4

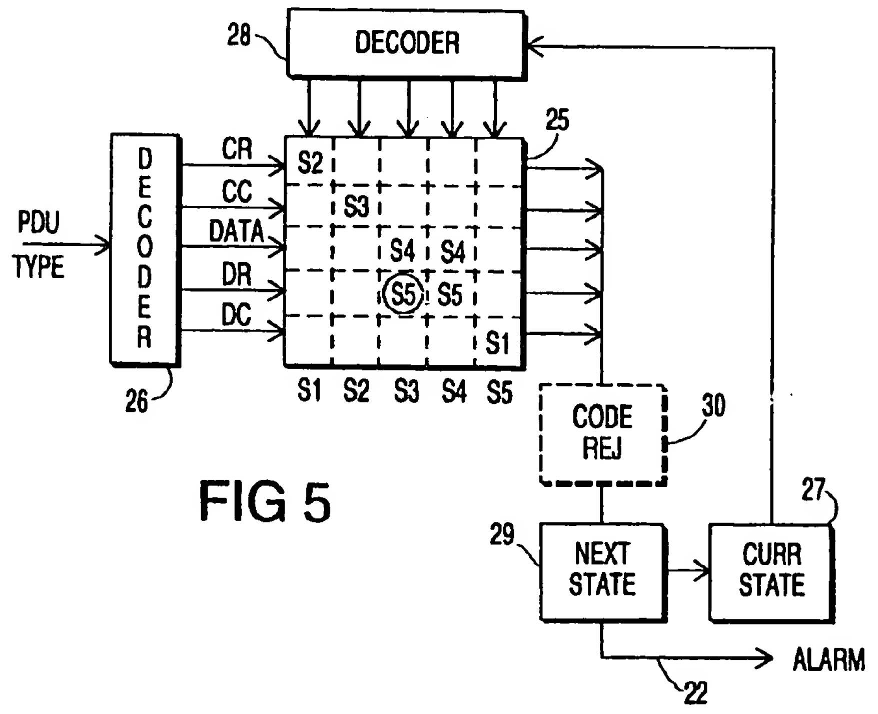
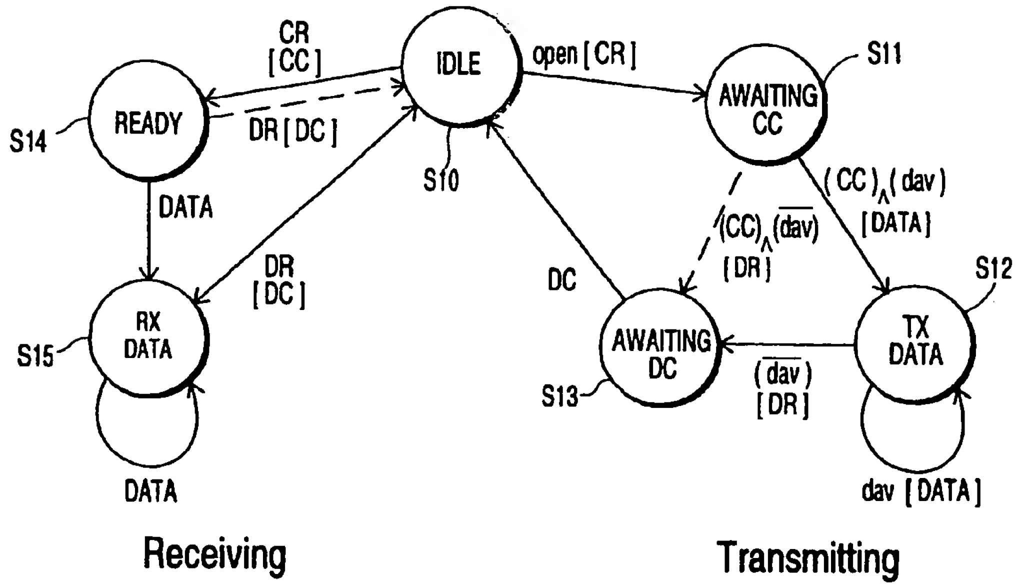

28
DECODER
25
DECODER
PDU TYPE
CR
CC
DATA
DR
DC
S2
S3
S4
S4
S5
S5
S1
S1 S2 S3 S4 S5
26
CODE REJ
30
27
29
NEXT STATE
CURR STATE
ALARM
22

FIG 5



IDLE
S10
READY
S14
RX DATA
S15
AWAITING CC
S11
TX DATA
S12
AWAITING DC
S13
CR [CC]
DR [DC]
open [CR]
DATA
DR [DC]
DC
(CC)∧(dav) [DATA]
(CC)∧(dav) [DR]
(dav) [DR]
dav [DATA]
DATA
Receiving
Transmitting

FIG 6

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 91 30 8276

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
A	US-A-4 748 348 (T. THONG) * column 2, line 11 - line 60 * * column 3, line 10 - column 4, line 37 * * figures 1,2 * ---	1	H04L12/26
A	COMPUTER DESIGN. vol. 23, no. 7, June 1984, LITTLETON, MASSACHUSETTS US pages 21 - 23; P. KILLMON: 'PROTOCOL ANALYZERS TAKE ON FUNCTIONS TO DEAL WITH NETWORK COMPLEXITIES' * page 22, middle column, line 5 - page 23, left column, line 31 * ---	1	
A	EP-A-0 332 286 (HEWLETT-PACKARD) * page 4, line 1 - line 31 * ---	1-10	
A	EP-A-0 371 656 (TEKTRONIX) * page 2, line 49 - page 3, line 8 * * page 3, line 55 - page 4, line 24 * * page 4, line 46 - line 58 * * claims 1,2; figures 1-3 * ---	1-10	TECHNICAL FIELDS SEARCHED (Int. Cl.5)
A	AT & T TECHNICAL JOURNAL. vol. 64, no. 10, December 1985, NEW YORK US pages 2413 - 2433; G.J. HOLZMANN: 'TRACING PROTOCOLS' * paragraph V * -----	1-10	H04L G06F H04Q

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 JANUARY 1992	CANOSA ARESTE C.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document